

Fig. 1

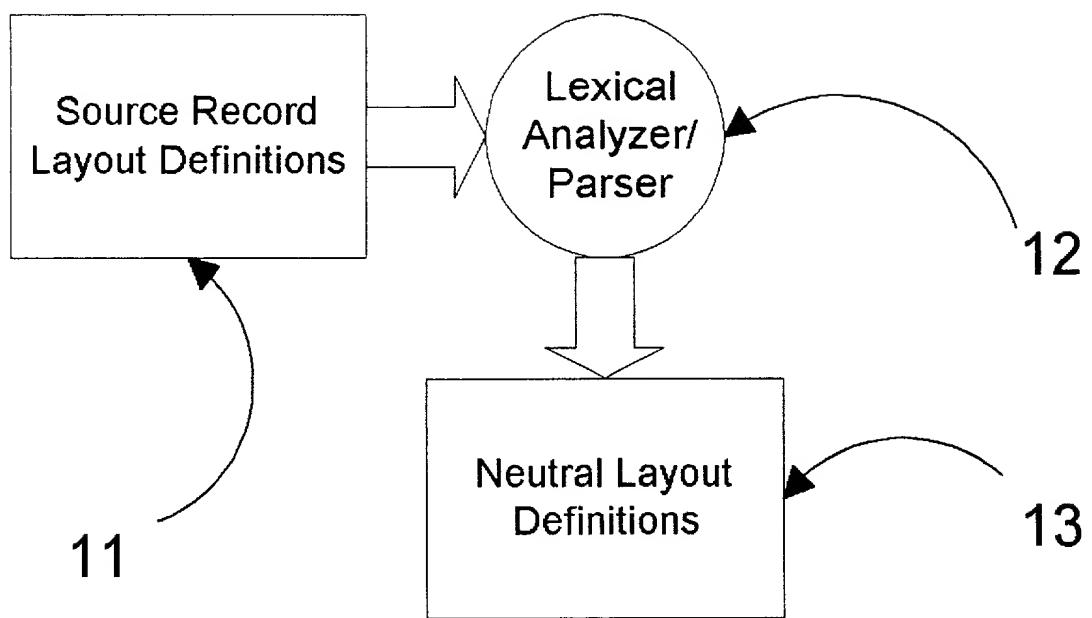


Fig. 2

01 STUDENT-SUMMARY-INFORMATION.
05 ID-NUMBER PIC 9(6).
05 PIN PIC 9(6).
05 NAME PIC A(35).
05 ADDRESS PIC A(25)
 OCCURS 3 TIMES.
05 PHONE-NUMBER PIC 999-999-9999.
05 SOCIAL-SECURITY-NUMBER PIC 999-99-9999.
05 GRADE-POINT-AVERAGE PIC 9V99.
05 BALANCES.
10 TUITION PIC S9(5) COMP-3.
10 HOUSING PIC S9(5) COMP-3.

Fig. 3

```

<?xml version="1.0"?>
<!DOCTYPE record SYSTEM "/XML/Meta/tmeta.dtd">
<record name="STUDENT-SUMMARY- INFORMATION" architecture="s390" align="1">
    <field type="pic" align="1" spec="999999" size="6">
        <name>ID-NUMBER</name>
        <association>ID-NUMBER</association>
    </field>
    <field type="pic" align="1" spec="999999" size="6">
        <name>PIN</name>
        <association>PIN</association>
    </field>
    <field type="pic" align="1" spec="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX" size="35">
        <name>NAME</name>
        <association>NAME</association>
    </field>
    <array size="3">
        <name>ADDRESS</name>
        <association>ADDRESS</association>
        <field type="pic" align="1" spec="XXXXXXXXXXXXXXXXXXXXXX" size="25">
            <name>ADDRESS</name>
            <association>ADDRESS</association>
        </field>
    </array>
    <field type="pic" align="1" spec="999X999X9999" size="12">
        <name>PHONE-NUMBER</name>
        <association>PHONE-NUMBER</association>
    </field>
    <field type="pic" align="1" spec="999X99X9999" size="11">
        <name>SOCIAL-SECURITY-NUMBER</name>
        <association>SOCIAL-SECURITY-NUMBER</association>
    </field>
    <field type="pic" align="1" spec="999" shift="-2" size="3">
        <name>GRADE-POINT-AVERAGE</name>
        <association>GRADE-POINT-AVERAGE</association>
    </field>
    <struct>
        <name>BALANCES</name>
        <association>BALANCES</association>
        <field type="packed" align="1" size="3">
            <name>TUITION</name>
            <association>TUITION</association>
        </field>
        <field type="packed" align="1" size="3">
            <name>HOUSING</name>
            <association>HOUSING</association>
        </field>
    </struct>
</record>

```

Fig. 4A

```

package com.touchnet.beangen;

import com.touchnet.base.*;
import java.io.*;
import java.util.*;

/**
 * This will provide the functionality that is common to all generated JavaBeans that
 * map into legacy structures
 *
 * Creation date: (12/14/99 1:28:08 PM)
 * @author: Gary Murphy
 */
public abstract class AbstractStructure
    implements StructureInterface
{
    private String             architecture;
    private StructTreeNode      root      = null;
    private BinaryRenderingEngine engine = new BinaryRenderingEngine();
    private java.lang.String metadataName;

    /**
     * Create the base constructure for Java objects that wrapper legacy data
     * structures
     */
    public AbstractStructure()
    {
        super();
    }

    /**
     * Access the name of the architecture that the underlying binary data
     * represents
     */
    public String getArchitecture()
        throws TException
    {
        return architecture;
    }

    /**
     * This will access an array within the structure. It will be returned as
     * an array of some concrete instance of this AbstractStructure. Even if
     * the array is of a single field, it will still be represented as a
     * structure that simply contains a single element. If the requested
     * element is not an array, this will throw an exception
     */
    public StructureInterface[] getArray(String name)
        throws TException
    {
        AbstractStructureTreeNode node = getNode(name);
        if (node instanceof ArrayTreeNode)
        {
            ArrayTreeNode arrayNode = (ArrayTreeNode)node;
            return arrayNode.getArray();
        }

        // If this isn't an array node, then we tried to access a non-array
        // as an array

        throw new TException("Attempt to access a non-array element as an array");
    }

    /**
     * Access the binary rendering engine.
     *
     * Creation date: (1/3/00 1:11:03 PM)
     * @return com.touchnet.base.BinaryRenderingEngine
     */
    protected BinaryRenderingEngine getEngine()
    {
        if (null == engine)
            engine = new BinaryRenderingEngine();
        return engine;
    }
}

```

Fig. 4B

```

    /**
     * Access the named field within the component
     */
    public String getField(String name)
        throws TException
    {
        AbstractStructureTreeNode node = getNode(name);
        if (node instanceof FieldTreeNode)
        {
            FieldTreeNode fieldNode = (FieldTreeNode)node;
            return fieldNode.getField().toString();
        }

        // It's not a field, so this is an exception
        throw new TException("Attempt to access a non-field element as a field");
    }

    /**
     * Access the name of the metadata that describes this component
     *
     * Creation date: (2/29/00 11:24:58 AM)
     * @return java.lang.String
     */
    public String getMetadataName()
    {
        return metadataName;
    }

    /**
     * This will access the named node, starting at the root of the embedded tree
     *
     * Creation date: (2/29/00 11:43:09 AM)
     * @return com.touchnet.beangen.AbstractStructureTreeNode
     * @param name java.lang.String
     * @exception com.touchnet.base.TException The exception description.
     */
    protected AbstractStructureTreeNode getNode(String name)
        throws TException
    {
        StringTokenizer tokenizer = new StringTokenizer(name, "/");
        return getNode(tokenizer,.getRoot());
    }

    /**
     * This will access the named node, as a child of the current node. The name
     * is the next element in the tokenizer. If the name child doesn't exist, this
     * will throw an exception
     *
     * Creation date: (2/29/00 11:43:09 AM)
     * @return com.touchnet.beangen.AbstractStructureTreeNode
     * @param name java.lang.String
     * @exception com.touchnet.base.TException The exception description.
     */
    protected AbstractStructureTreeNode
        getNode(StringTokenizer tokenizer, AbstractStructureTreeNode current)
        throws TException
    {
        if (!tokenizer.hasMoreElements())
            return current; // The current node is the requested node

        String child = tokenizer.nextToken();

        // Look for the name among the child nodes
        int count = current.getChildCount();
        for (int i = 0; i < count; ++i)
        {
            AbstractStructureTreeNode node =
                (AbstractStructureTreeNode)current.getChildAt(i);
            if (node.getName().equals(child))
                return getNode(tokenizer, node);
        }
    }

```

Fig. 4C

```

// The name didn't match any of the children

        throw new TException("The child of '"+current.getName()+"' named '"+child+"'
                                does not exist");
    }

    /**
     * This will access the root node for the legacy data layout
     *
     * Creation date: (1/3/00 12:56:48 PM)
     * @return com.touchnet.beangen.StructTreeNode
     */
    protected StructTreeNode getRoot()
    {
        return root;
    }

    /**
     * This will read the binary contents of the input stream and
     * place it in the appropriate nodes of the tree
     */
    public void read(InputStream stream)
        throws TException
    {
        // Code not shown
    }

    /**
     * Access the name of the architecture that describes the underlying
     * binary data.
     */
    public void setArchitecture(String name)
        throws TException
    {
        architecture = name;
        return;
    }

    /**
     * Set the array for this level in the data structure
     */
    public void setArray(String name, StructureInterface[] child)
        throws TException
    {
        AbstractStructureTreeNode node = getNode(name);
        if (node instanceof ArrayTreeNode)
        {
            ArrayTreeNode arrayNode = (ArrayTreeNode)node;
            arrayNode.setArray(child);
        }

        // If this isn't an array node, then we tried to access a non-array
        // as an array

        throw new TException("Attempt to access a non-array element as an array");
    }

    /**
     * Update the named field with the value
     */
    public void setField(String name, String value)
        throws TException
    {
        AbstractStructureTreeNode node = getNode(name);
        if (node instanceof FieldTreeNode)
        {
            FieldTreeNode fieldNode = (FieldTreeNode)node;
            LegacyField field = fieldNode.getField();
            field.setValue(value);
        }

        // It's not a field, so this is an exception

        throw new TException("Attempt to access a non-field element as a field");
    }
}

```

Fig. 4D

```
        }

    /**
     * Access the name of the metadata that describes this component
     *
     * Creation date: (2/29/00 11:24:58 AM)
     * @param name java.lang.String
     */
    public void setMetadataName(String name)
    {
        metadataName = name;
        return;
    }

    /**
     * This will access the root node for the legacy data layout
     *
     * Creation date: (1/3/00 12:56:48 PM)
     * @param rootNode com.touchnet.beangen.StructTreeNode
     */
    protected void setRoot(StructTreeNode rootNode)
    {
        root = rootNode;
        return;
    }

    /**
     * This will write the binary contents back to the
     */
    public void write(OutputStream stream)
        throws TException
    {
        // Code not shown
    }
}
```

Fig. 5A

```

package com.touchnet.beangen.generated;

import com.touchnet.beangen.*;
import com.touchnet.base.*;
/**
 * This was automatically generated 2/29/00 12:38:47 PM
 */
public class StudentSummaryInformation
    extends AbstractStructure
{
/**
 * StudentSummaryInformation constructor comment.
 */
public StudentSummaryInformation() {
    super();
}
public String getAddress(int index)
    throws TException
{
    StructureInterface[] array = getArray("/ADDRESS");
    return array[index].getField("/");
}
public String getGradePointAverage()
    throws TException
{
    return getField("/GRADE-POINT-AVERAGE");
}
public String getHousing()
    throws TException
{
    return getField("/BALANCES/HOUSING");
}
public String getIdNumber()
    throws TException
{
    return getField("/ID-NUMBER");
}
public String getName()
    throws TException
{
    return getField("/NAME");
}
public String getPhoneNumber()
    throws TException
{
    return getField("/PHONE-NUMBER");
}
public String getPIN()
    throws TException
{
    return getField("/PIN");
}
public String getSocialSecurityNumber()
    throws TException
{
    return getField("/SOCIAL-SECURITY-NUMBER");
}
public String getTuition()
    throws TException
{
    return getField("/BALANCES/TUITION");
}
public void setAddress(int nth, String value)
    throws TException
{
    StructureInterface[] array = getArray("/ADDRESS");
    array[nth].setField("/", value);
}
public void setGradePointAverage(String value)
    throws TException
{
}

```

Fig. 5B

```
        setField("/GRADE-POINT-AVERAGE",value);
    }
    public void setHousing(String value)
    throws TException
    {
        setField("/BALANCES/HOUSING",value);
    }
    public void setIdNumber(String value)
    throws TException
    {
        setField("/ID-NUMBER",value);
    }
    public void setName(String value)
    throws TException
    {
        setField("/NAME",value);
    }
    public void setPhoneNumber(String value)
    throws TException
    {
        setField("/PHONE-NUMBER",value);
    }
    public void setPIN(String value)
    throws TException
    {
        setField("/PIN",value);
    }
    public void setSocialSecurityNumber(String value)
    throws TException
    {
        setField("/SOCIAL-SECURITY-NUMBER",value);
    }
    public void setTuition(String value)
    throws TException
    {
        setField("/BALANCES/TUITION",value);
    }
}
```

Fig. 6

Fig. 7

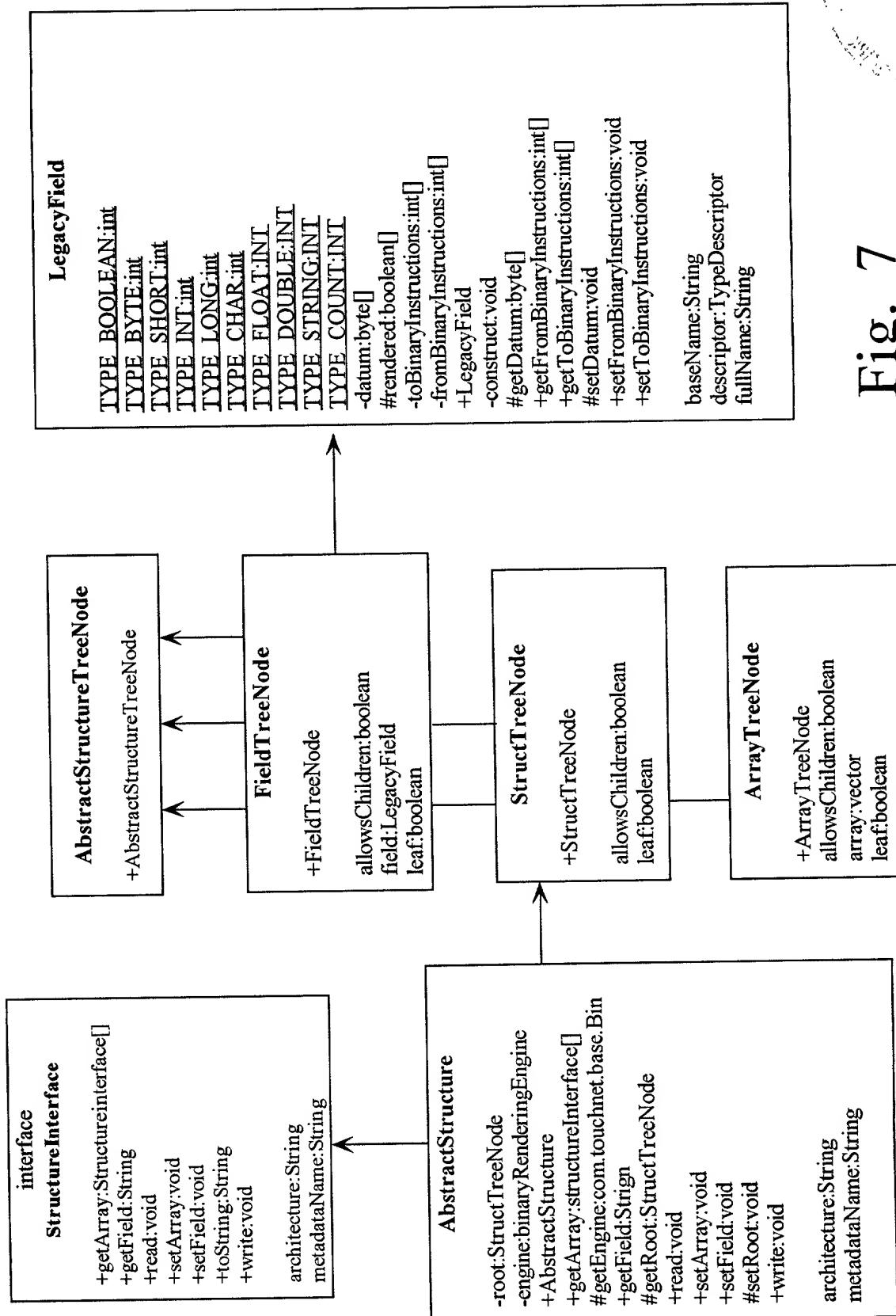
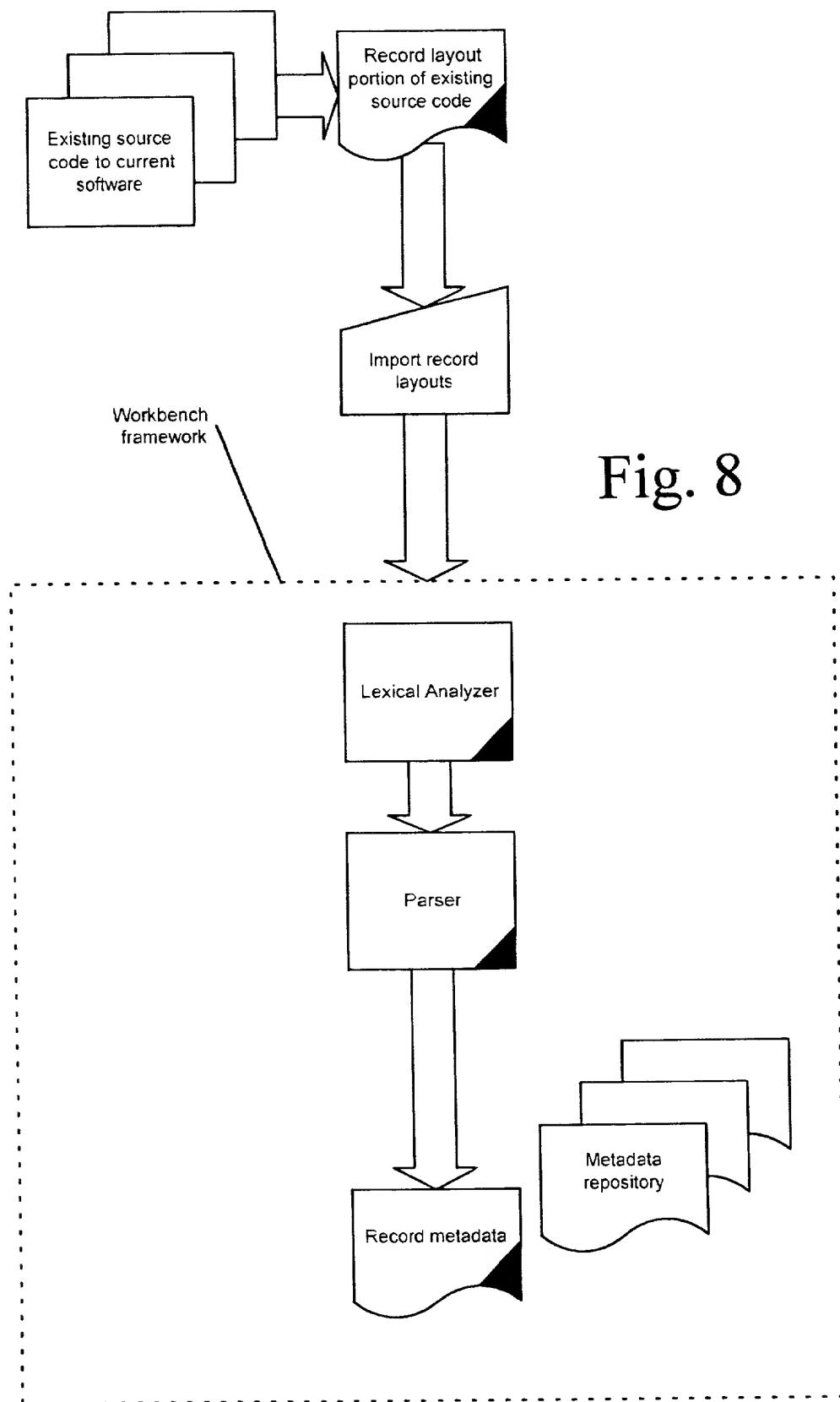


Fig. 8



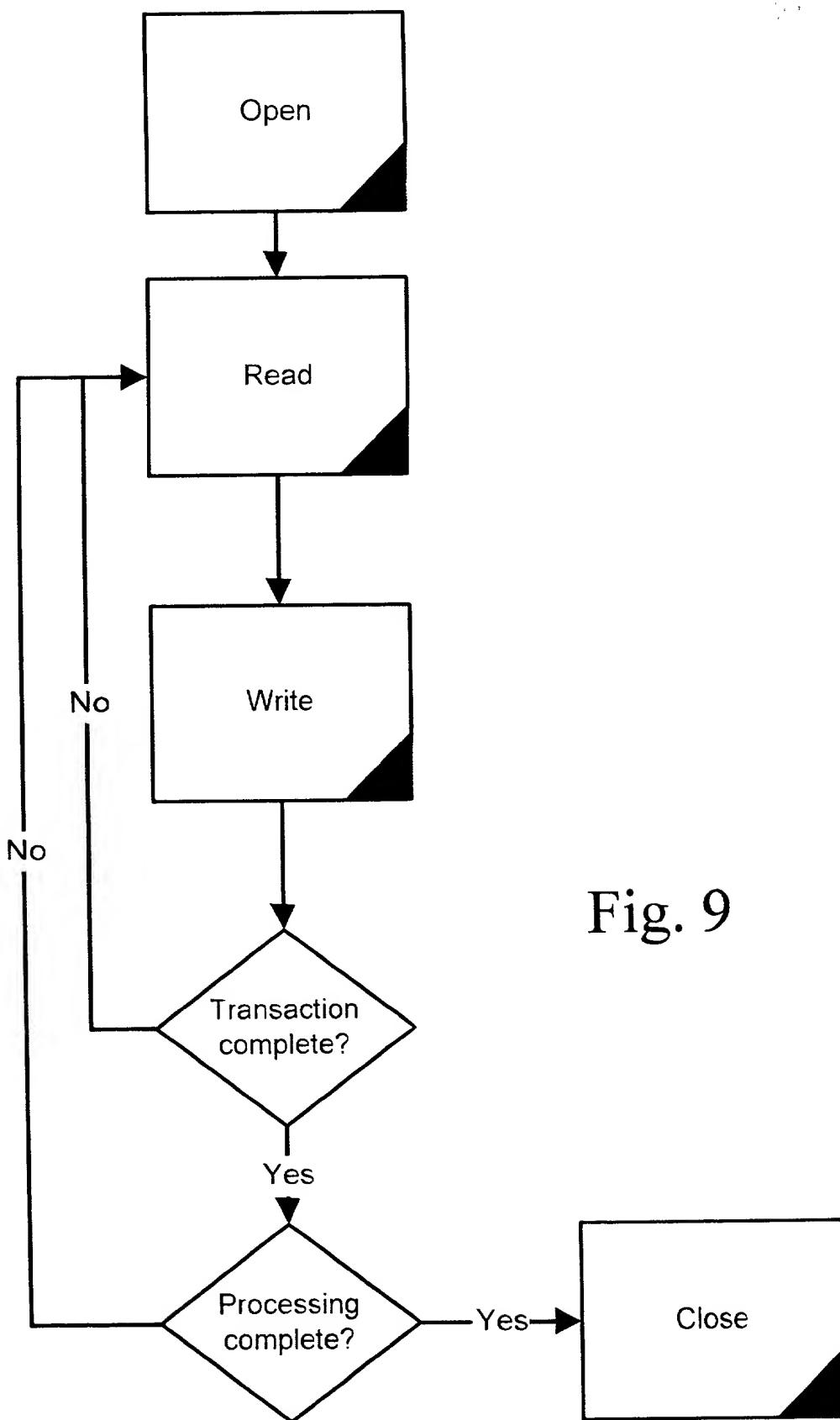


Fig. 9

```
/* -- Published APIs and data types */

typedef long lxsHandle;

lxsHandle lxsOpen(char *ic, char *host,
                  unsigned short port);
in:    lxsClose(lxsHandle handle);
in:    lxsRead(lxsHandle handle, char *name, void *data, unsigned long length);
in:    lxsWrite(lxsHandle handle, char *name, void *data, unsigned long length),
in:    lxsCommit(lxsHandle handle);
in:    lxsRollback(lxshandle handle);
vcic   lxsGetLastName(lxsHandle handle, char *name),
```

Fig. 10

Fig. 11

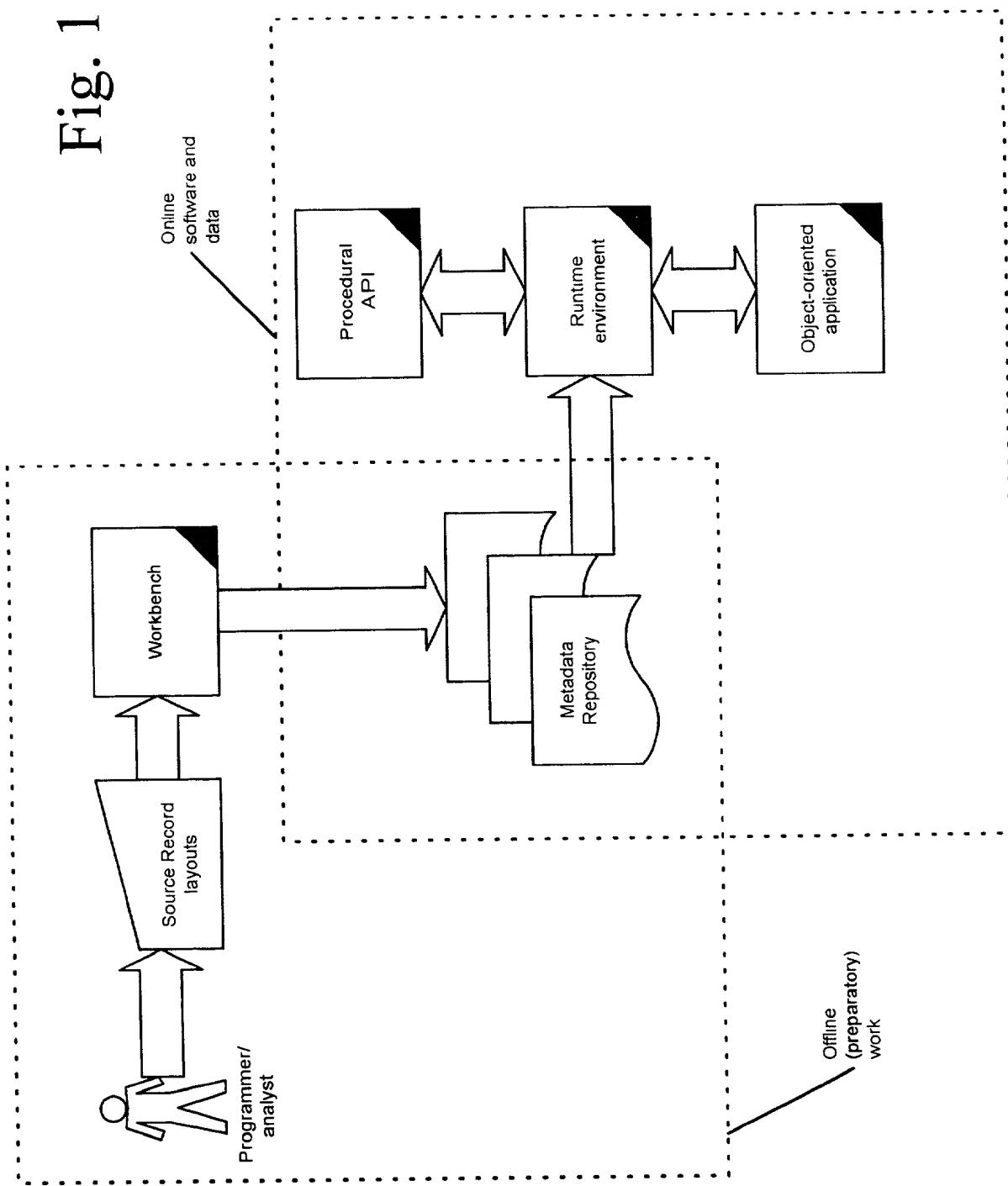
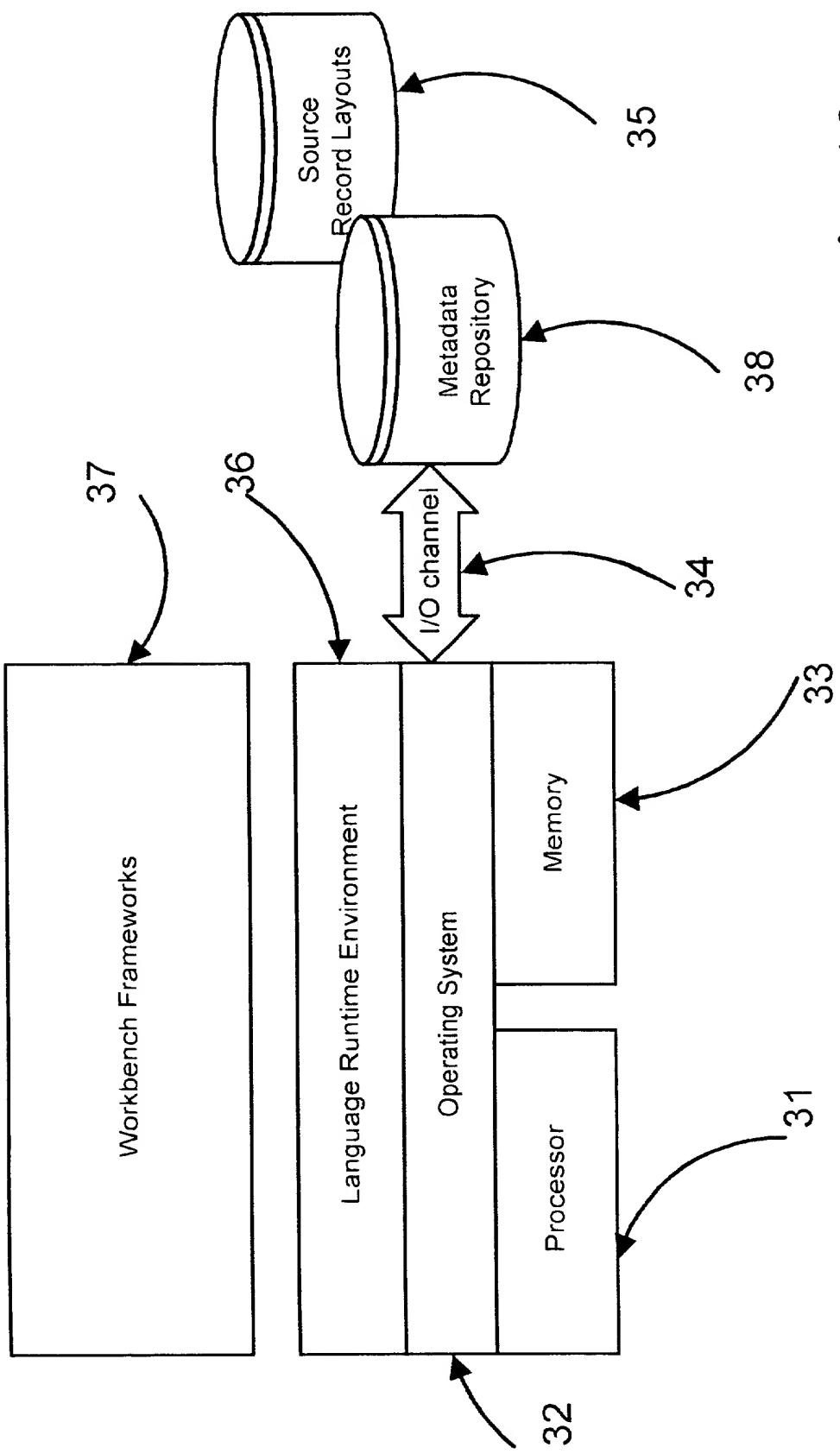


Fig. 12



```

package com.touchnet.util.base;

*****<copyright>*****
/*
 * Copyright (c) 2000
 * TouchNet Information Systems, Inc.
 * All Rights Reserved
 */
/*
 * This program is an unpublished copyright work of TouchNet Information
 * Systems, Inc. of Lenexa, KS. The program, all information disclosed
 * and the matter shown and described hereon or herewith are confidential
 * and proprietary to TouchNet Information Systems, Inc.
*/
*****<copyright>*****
/*
 * Change Log.
 * $Log: BinaryRenderingEngine.java $
 * Revision 1.4 2000/07/19 10:36:38 gln
 */
import com.touchnet.util.base.*;
import com.touchnet.util.*;
import java.math.BigInteger;
/**
 * This is a utility object that will manage the bit/byte manipulation
 * for a variety of data conversions.
 */
public class BinaryRenderingEngine
{
    /**
     * Construct an object that will render byte arrays in a variety
     * of formats
     */
    public BinaryRenderingEngine()
    {
        super();
    }
    /**
     * Access the value that is used when there is a rendering error
     */
    public byte getErrorByte()
    {
        return errorByte;
    }
    /**
     * Return a copy of one of these.
     */
    public COM.touchnet.xmlhost.BinaryRenderingEngine
    getInstance()
    {
        if (instance == null)
            instance = new BinaryRenderingEngine();
        return instance;
    }
    /**
     * This is called when there is a formatting exception such as a
     * string representation of a number that overflows the number of
     * bytes that number can handle
     */
    public void handleFormatException(byte[] data, IllegalArgumentException exception)
    {
        // For now, we just set the bytes to some pre-defined value. We may want
        // to make this a JavaBean that fires an formatting exception event to
        // the listeners.
    }
}

```

Fig. 13A

```

byte err = getErrorByte();
for (int i = 0; i < data.length; ++i)
    data[i] = err;
return;

/***
 * This will parse the string into a long
 *
 * Creation date: (7/12/00 11:21:57 AM
 * @return long
 * @param number java.lang.String
 */
private long parseLong(String number
{
    if (0 == number.length())
        return 0;

    // The Java parseLong() is pretty stupid. It can't handle a leading '+', so I need
    // an explicit check for that.

    if ('+' == number.charAt(0))
        number = number.substring(1);

    return Long.parseLong(number);
}
/***
 * Render a Java String from a series of bytes with 7-bit ASCII values
 *
 * @return java.lang.String
 * @param datum byte[]
 */
public String renderAsciiString(byte[] datum)
{
    int size = datum.length;
    char[] array = new char[size];
    for (int i = 0; i < size; ++i)
        array[i] = (char)renderPrintableAscii(datum[i], ' ');

    return String.valueOf(array);
}
/***
 * This will return a byte array containing 7-bit ASCII values generated
 * from the number passed
 *
 * @return byte[]
 * @param value int
 * @param size int
 * @param pad char
 */
public byte[] renderAsciiString(int value, int size, char pac
{
    byte[] buffer = new byte[size];
    int offset = 0;
    boolean negative = false;

    if ((value < 0) && (pad != ' '))
    {
        value = 0 - value;
        negative = true;
        buffer[offset++] = (byte)'-';
    }

    String string = Integer.toString(value);
    int length = string.length();
    for (; offset < size - length; ++offset)
        buffer[offset] = (byte)pac; // Pad on left if needed

    byte[] stringBytes = string.getBytes();
    for (int i = 0; offset < size; ++offset, ++i)
        buffer[offset] = stringBytes[i];
}

```

Fig. 13B

```

        return buffer;
    }

    /**
     * This will render the two bytes in the array into an
     * integer and return the string rendering of that.
     *
     * @return java.lang.String
     * @param raw byte[]
     */
    public String renderBigEndian16Bit(byte[] raw
    {
        short byte0 = (short)raw[0]; // Allow this to sign-extend
        short byte1 = (short)(raw[1] & 0x00FF);

        short value =
            (short)((byte0 << 8
                    - byte1
                    );
        return String.valueOf(value);
    }

    /**
     * This will render the string numeric into two bytes.
     *
     * @param java.lang.String
     * @return raw byte[]
     */
    public byte[] renderBigEndian16Bit(String datum,
    {
        byte[] raw = new byte[2];
        short value = 0;
        try
        {
            value = parseShort(datum);
            raw[0] = (byte)((value & 0x0000FF00) >> 8);
            raw[1] = (byte)( value & 0x000000FF);
        }
        catch(NumberFormatException exception)
        {
            handleFormatException(raw, exception);
        }
        return raw;
    }
}

```

Fig. 13C

```

    /**
     * This will take a series of bytes which are expected to be
     * ASCII characters representing numbers, for example:
     *
     * { '-' , '6' , '9' , '6' , '0' }
     *
     * would be -6960. It will return an int
     *
     * @return int
     * @param raw byte[]
     */
    public int renderIntegerFromAsciiBytes(byte[] raw
    {
        String number = renderAsciiZString(raw).trim();

        if ("".equals(number)) // All white space is considered a valid zero integer
            return 0;

        int value = 0;
        try
        {

```

```

        value = parseInt(number);
    }
    catch(NumberFormatException exception)
    {
        handleFormatException(raw, new NumberFormatException());
        return -1;
    }
    return value;
}

.

.

.

/***
 * This will render bytes representing a packed decimal field into
 * a string representation.  This is a helper routine that works
 * for both signed and unsigned packed values
 *
 * @return java.lang.String
 * @param raw byte[]
 * @param isSigned boolean
 */
private String renderPacked(byte[] raw, boolean isSigned, int offset)
{
    char      signCharacter = ' '; // Assume no sign.
    StringBuffer buffer      = new StringBuffer();
    boolean   minus         = false;

    // Take a peek at the offset compared to the length of the raw data and see
    // where the decimal point goes.

    int append   = 0;
    int insertAfter = -1;
    int digits   = (raw.length << 1) - 1;

    if (offset > 0) // Append only
        append = offset;
    else
    {
        // We have a negative offset, the decimal will either be to the left or
        // somewhere in the middle.

        insertAfter = digits + offset; // Add because offset is negative
        if (insertAfter < 0) // The offset means only leading zeros...
        {
            buffer.append('.');
            for (int i = insertAfter; i < 0; ++i)
                buffer.append('0');
        }
    } // else

    int     rIndex      = -1; // Index into the raw data
    int     nibble      = 0;
    boolean secondNibble = true;

    for (int i = 0; i < digits; ++i)
    {
        if (secondNibble) // Bump input byte every other nibble
            ++rIndex;
        secondNibble = !secondNibble;

        // Wait for the iteration in which we have to stuff the extra decimal
        // point.

        if (i == insertAfter)
            buffer.append('.');
        if (secondNibble)
            nibble = raw[rIndex] & 0x0000000F;
    }
}

```

Fig. 13D

Fig. 13E

```
        else
            nibble = (raw[rIndex] >> 4) & 0x0000000F;

        switch(nibble)
        {
            case 0: buffer.append('0'); break;
            case 1: buffer.append('1'); break;
            case 2: buffer.append('2'); break;
            case 3: buffer.append('3'); break;
            case 4: buffer.append('4'); break;
            case 5: buffer.append('5'); break;
            case 6: buffer.append('6'); break;
            case 7: buffer.append('7'); break;
            case 8: buffer.append('8'); break;
            case 9: buffer.append('9'); break;
            default:
                handleFormatException(raw,
                    new IllegalArgumentException("Invalid value in data"));
                return "[data format error]";
        } // switch.
    } // for

    // Now handle the last nibble which is the sign.

nibble = raw[rIndex] & 0x0000000F;
switch(nibble)
{
    case 0xA:
    case 0xC:
    case 0xE:
    case 0xF:
        break;
    case 0xD:
    case 0xB:
        minus = true;
        break;
    default:
        {
            handleFormatException(raw,
                new IllegalArgumentException("Invalid value in data"));
            return "[data format error]";
        }
}

// Append any additional trailing zeros that are a result of the decimal shift
// in the type descriptor:

for (int i = 0; i < append; ++i)
    buffer.append('0');

String rendered = buffer.toString();
if (isSigned && minus)
    rendered = '-' + rendered;
return rendered;
}

/*
 * This is a helper method that will render PIC templates that have been pre-determined
 * to be numeric. It will handle both EBCDIC or ASCII input numerics.
 *
 * @return byte[]
 * @param raw java.lang.String
 * @param template byte[]
 * @param offset int
 * @param isAscii boolean
 */
private byte[] renderPacked(String raw, int size, int offset, boolean isSigned)
{
    byte[]      buffer  = new byte[size],
    int         shift   = 0; // This is the decimal place shift that we find in the
                           // data. It is used to reconcile the offset param
    boolean     decimal = false; // ... until we hit a decimal point, then it is true
```

```

boolean      minus   = false;
byte[] userdata = raw.getBytes();
byte[] numeric  = new byte[userdata.length]; // Just the numeric part of the data
int    numSize   = 0; // Count of just the numerics in the user data

for(int i = 0; i < userdata.length; ++i)
{
    switch(userdata[i])
    {
        case (byte)'0':
        case (byte)'1':
        case (byte)'2':
        case (byte)'3':
        case (byte)'4':
        case (byte)'5':
        case (byte)'6':
        case (byte)'7':
        case (byte)'8':
        case (byte)'9':
            numeric[numSize++] = (userdata[i]);
            if (decimal) ++shift;
            break;

        case (byte)'-':
            minus = true;
            break;
        case (byte)'+':
            break;
        case (byte)'.':
            decimal = true;
            break;
    } // switch
} // for

// Now we have the digits separated from the sign and decimal point. Now
// we have to normalize the decimal offset and the digit count with the
// template. What makes this additionally complex is the observation that
// there can be truncation on either side of the user data if the shift
// overflows the template. Consider the following examples:
//
// Assume:
//
//     template = 99999 with shift -2 (via PIC 999V99)
//
//     Userdata      Answer
//     -----
//     1230          23000 (truncation on left)
//     123           12300
//     12.3          01230
//     1.23          00123
//     .123          00012 (truncation on right)
//
//     At this point in the code, we have the user data filtered out
//     into a the string "123". We need to align the decimal point
//     logically based on the shifts in the template minus the logical
//     shifts from the explicit decimal point in the data.

int    index  = numSize - ((size << 1) - 1) - offset - shift;
int[] value   = new int[2];
for (int i = 0; i < size-1; ++i)
{
    for (int j = 0; j < 2; ++j)
    {
        if (index < 0)
            value[j] = 0;
        else
            if (index < numSize)
                value[j] = numeric[index] & 0x0000000F;
            else
                value[j] = 0;
        ++index;
    }
}

```

Fig. 13F

```
        }
        buffer[1] = (byte)((value[0] << 4) | value[1]);
    }

// Do the last byte as a special case since it contains the sign nibble

for (int j = 0; j < 2; ++j)
{
    if (index < 0,
        value[j] = 0;
    else
        if (index < numSize)
            value[j] = numeric[index] & 0x0000000F;
        else
            value[j] = 0;
    ++index;
}
int sign = 0x0C; // Plus
if (isSigned && minus)
    sign = 0xD;
buffer[size-1] = (byte)((value[0] << 4) | sign);
return buffer;
}
```

Fig. 13G